Lab Exercise 3: Working with Doubly Linked List and ADTs

SE2205: Data Structures and Algorithms using Java – Fall 2022

Open Day: October 14, 2022; **Cut off time**: Wednesday October 19, 2022 @11pm

Prepared by Dr. Quazi Rahman (qrahman3@uwo.ca).

A. Rationale and Background

In this lab, in the first exercise you will work on Doubly linked list whose codes are given in the course handout and added here (as Appendix – page 7 and 8) too. Also, you need to work on the concepts of ‘natural ordering’ and ‘order by comparator’ in a second exercise.

B. Evaluation and Submission Instructions

You will get credit for this lab exercise when you submit the working code. No part-mark will be awarded if the code does not run. No mark will be awarded if you submit your code with a bunch of print-statements. Submit your lab online after carrying out the following instructions:

1. Create a Project with the following name: *username_LabExercise3*
2. While working on question 1, create a package LE3Q1. Create a package LE3Q2 when working on question 2.
3. You are required to use the same identifiers as outlined in the UML diagrams, and in the questions.
4. For each question, use the static header and footer methods your created before, and then modify those according to the requirements outlined in the questions.
5. Comments: Writing comment for Lab Exercises is not mandatory but it is recommended.
6. Once the assignment is completed, go to your ‘Assignments’ folder. Select the project folder (e.g., *username_LabExercise3*). Right-click to select ‘Send to’ ‘Compressed zipped folder’. Make sure it has the same naming convention (e.g., *username_LabExercise3.zip*). Upload this file to OWL as your submission.
7. If you have any question on the lab exercise, please see your instructor either during his office hours or at the end of the lecture. Also, you can see your TAs after making an appointment during the tutorial hours.

C. Lab Question

1. [15 Marks]

Working with Doubly Linked List. Here you are given the DoubyLinkedLIst Class [See the Appendix], which you need to modify by adding one public method findNode() and overriding the public toString() method to print the doubly linked list. Please study the given code before you add the required methods. Your good strategy would be to work on the toString() method first and once that is successful, work on the findNode() method. Also, you need to create a class called Student with two private fields firstName and score. Finally, you need to create a driver class called *Driver_ DoubyLinkedLIst* in which the driver method will check your code according to the sample output.

a) Define a class called *Student* with the following specifications (see the class UML diagram below):

- Two private data fields: firstName and score.
- Constructor without parameter: This should be defined based on your own first name and any score of your choice.
- Constructor with parameter: Define it according to the convention.
- Override toString() method that will return a string containing the first name, followed by a colon (:), and finally followed by the Score value up to two decimal places. E.g., Quazi: 89.50. Please note that the score should be displayed with up to two decimal places. Hint: Here, your String.format() method will come handy. Talk to the instructor during his office hours or after the lecture if you are not sure how to do it.

b) Add the following methods to the given DoublyLinkedList Class.

i) public Node<E> findNode (E e): This method will return the Node that contains the element e, or it returns null if the list is empty.

ii) public String toString(): Override the toString() method so that it prints the content of the list. Hint: Create an ArrayList<E> reference variable and instantiate it. Now, from the **header** to the **trailer** of the linked list traverse all the nodes in a loop and add each element from the corresponding node to the ArrayList<E>. Finally return the arrayListName.toString() from this method. By the way, you can follow any other approach.

c) Copy the Header method from the previous lab and then change the 'goal' according to the question. At the same time change the method header as given below, where labE_number is the Lab Exercise number and q_number is the question number received by this method:

```
public static void myHeader(int labE_number, int q_number)
```

d) Copy the Footer method from the previous lab and change the method header as given below, where labE_number is the Lab Exercise number and q_number is the question number received by this method:

```
public static void myFooter(int labE_number, int q_number)
```

e) Define the driver method and do the following:

i) Call the header method.

ii) Create a DoublyLikedList< > reference variable of MyStudent type called *yourFirstName*List and instantiate it with no-arg constructor.

iii) Create five MyStudent-type reference variables s0, s1, s2, s3 and s4, and instantiate those with the following values: (no argument constructor), (Harry, 67.35), (Luna, 87.5), (Vincent, 60.5), (Hermione, 89.2).

iv) Add first four students s0, s1, s2, s3 to the linked list using the addLast() method.

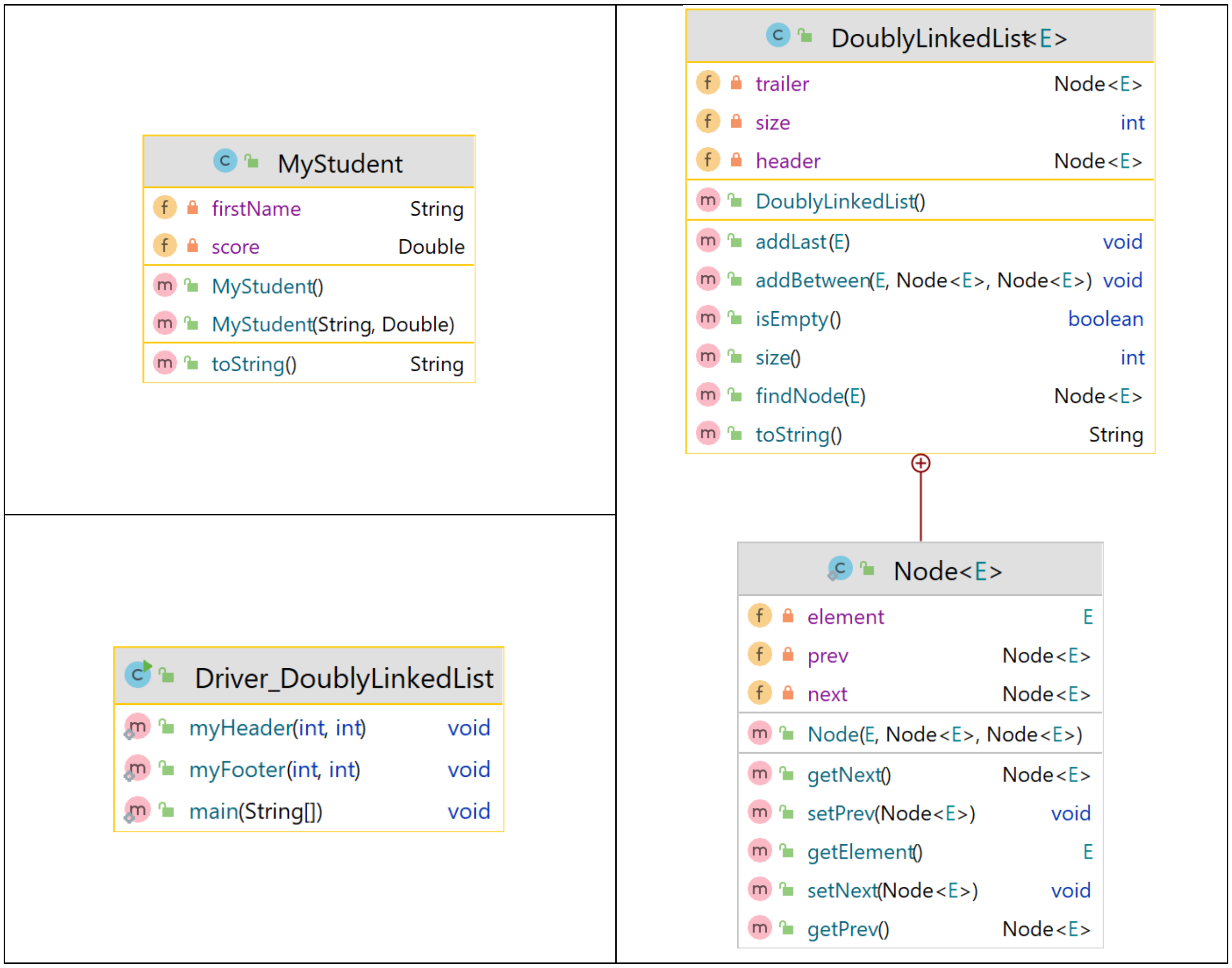
v) Print the list using toString() method.

vi) Find the node info for s2 and then for s3 using findNode() method. Hint: Since Node<> class has been declared as a nested class of DoublyLinkedList<> Class, the following statement must be used to create a reference variable of MyStudent type Node<> class:
`DoublyLinkedList.Node<MyStudent> anyName;`

vii) Now using the node info gathered from part (vi) add s4 in between s2 and s3 using addBetween() method.

viii) Print the list using toString() method again.

ix) Call the footer method.



Sample output:

```
=========================================================
Lab Exercise 3-Q1
Prepared By: YourFirstName YourLastName
Student Number: 999999999
Goal of this Exercise: ........!
=========================================================
Adding 4 students to the list.
The list Content:
[yourFirstName: 89.55, Harry: 67.50, Luna: 87.50, Vincent: 60.50]
Adding Hermione to the list in between Luna and Vincent.....
The list Content:
[yourFirstName: 89.55, Harry: 67.50, Luna: 87.50, Hermione: 89.20, Vincent: 60.50]
=========================================================
Completion of Lab Exercise 3-Q1 is successful!
Signing off - YourFirstName
=========================================================
```

2. [15 Marks]

In this exercise we will demonstrate our understanding on using Comparator<> Interface, Comparable<> Interface and Collections<> class with the help of a class called Student and a couple of Helper classes. Here are the requirements of this lab Exercise.

a) Define a class called *Student* that will implement the Comparable<T> Interface so that the Student's score can be compared naturally (natural order). The *Student* class will address the following specifications (see the class UML diagram below); you can copy the content of the MyStudent Class you created in Question 1, and add the other required functionalities:

- Three private data fields: firstName, lastName and score.
- Constructor without parameter: This should be defined based on your own first name, last name, and any score of your choice.
- Constructor with parameter: Define it according to the convention.
- Override *toString()* method that will return a string containing the first name, followed by the last name, followed by a colon (:), and finally followed by the Score value up to two decimal places. E.g., Quazi Rahman: 89.50. Please check the sample output when you write this definition. When writing this definition, in the first step, please keep it simple. Once everything runs successfully, format it according to the sample output.
- Override compareTo() method that should compare student's score. Note: We have discussed this in the class, and associated codes are available in the lecture handout.

b) Define a helper class called *HelperClassCompareFirstNames* that will implement the Comparator<T> Interface so that the Student's First names can be compared according to the 'order by comparator'. Here you need to implement the compare() method only. Note: We have discussed this in the class, and some associated codes are available in the lecture handout.

c) Define a second helper class called *HelperClassCompareLastNames* that will implement the Comparator<T> Interface so that the Student's Last names can be compared according to the 'order by comparator'. Here you need to implement the compare() method only. Note: We have discussed this in the class, and some associated codes are available in the lecture handout.

d) Create a driver class *DriverForStudentClass*, and define the following methods in there:

- The Header Method: Just copy it from Question 1 and modify it with the following method-header:
  i. `public static void myHeader(Student myInfo, int labE_number, int q_number)`
     When called, this method will display the header information on the screen with the help of Student reference variable myInfo for your full name, int type variable labE_number for lab-exercise number, and int type variable q_number for question number. The rest will be hardcoded as you did before.
- The footer Method: Just copy it from Question 1 re-use it as is.
- Driver method that will address the following specifications:
  i. Call the header method with appropriate arguments.
  ii. Create an ArrayList<> of Students and instantiate it with no-argument constructor.
  iii. Add a Student object to the above list using no-argument constructor.
  iv. Now add the following students to the list in sequence: (Harry Potter 75.5), (Ronald Weasley 86.0), (Hermione Granger 91.7), (Parvati Patil 93.75).

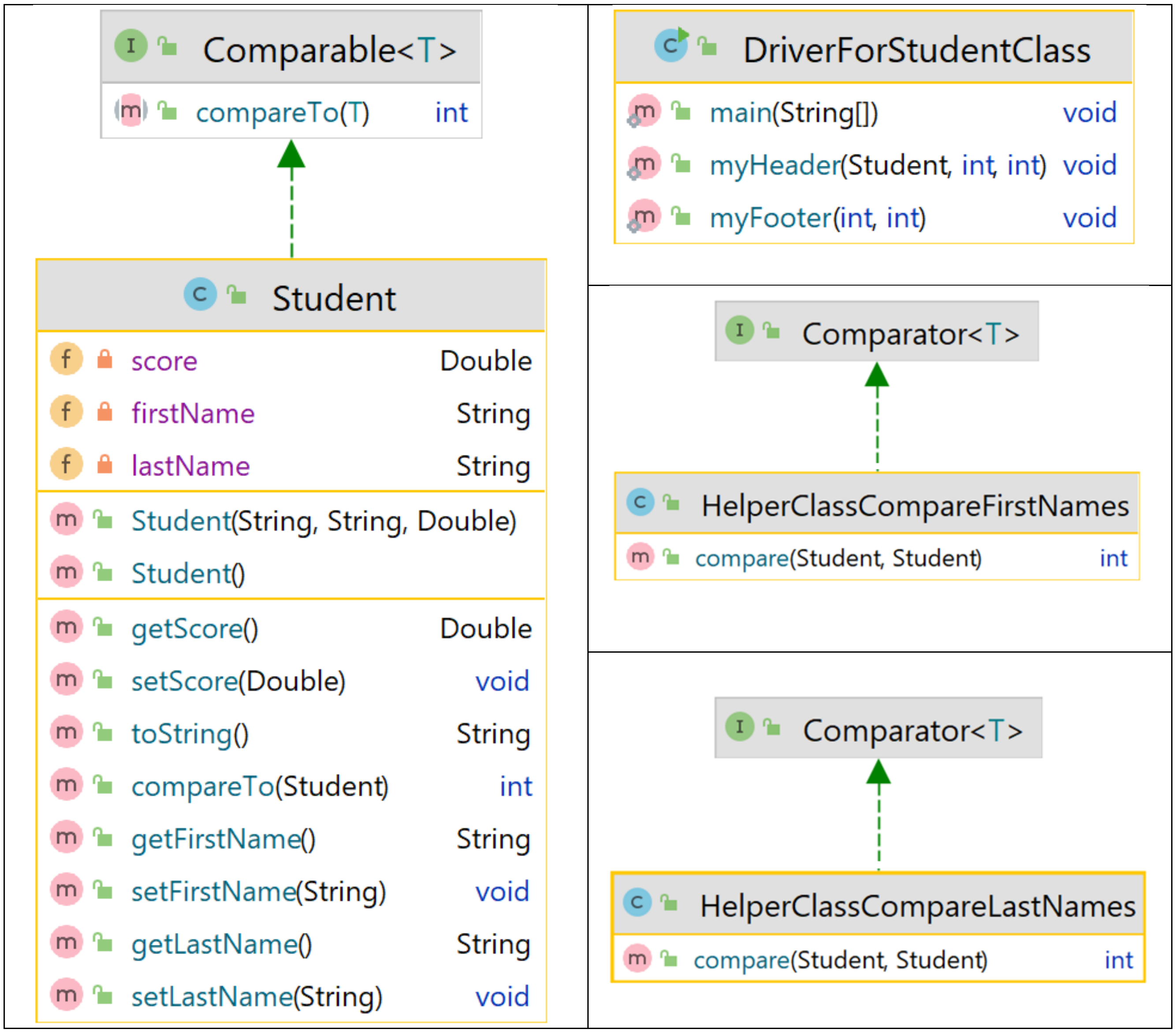
v. Print the score-card by calling the toString() method of the ArrayList<>. The output needs to follow the same format as shown in the sample output.

vi. Now using *Collections*'s sort method display the score card in descending order in terms of scores. (See the UML Diagram of the *Collections* class, available in Unit 2, Part 3 Handout). In this case, you will take the advantage of natural ordering by using the first sort method outlined in the UML diagram. Note that all the methods are static methods in the UML diagram there.

vii. Now using *Collections*'s sort method and a *HelperClassCompareLastNames* reference variable (See the UML Diagram of the *Collections* class, available in Unit 2, Part 3 Handout), sort the ArrayList<> and then display the sorted list in ascending order in terms of the last names.

viii. Now using *Collections*'s sort method and a *HelperClassCompareFirstNames* reference variable (See the UML Diagram of the *Collections* class, available in Unit 2, Part 3 Handout), sort the ArrayList<> and then display the sorted list in ascending order in terms of the first names.

ix. Call the footer method with appropriate arguments.



Sample output:

```
=====================================================
Lab Exercise 3-Q2
Prepared By: YourFirstname YourLastName
Student Number: 999999999
Goal of this Exercise: .....
=====================================================
The Score Card:
        YourFirstname YourLastName: 100.00
        Harry Potter: 75.50
        Ronald Weasley: 86.00
        Hermione Granger: 91.70
        Parvati Patil: 93.75

The sorted list in terms of score in descending order....
        YourFirstname YourLastName: 100.00
        Parvati Patil: 93.75
        Hermione Granger: 91.70
        Ronald Weasley: 86.00
        Harry Potter: 75.50

The sorted list in terms of Last Names....
        Hermione Granger: 91.70
        Parvati Patil: 93.75
        Harry Potter: 75.50
        Ronald Weasley: 86.00
        YourFirstname YourLastName: 100.00

The sorted list in terms of First Names....
        Harry Potter: 75.50
        Hermione Granger: 91.70
        Parvati Patil: 93.75
        Ronald Weasley: 86.00
        YourFirstname YourLastName: 100.00

=====================================================
Completion of Lab Exercise 3-Q2 is successful!
Signing off - YourFirstName
=====================================================
```

Appendix:

The given Code for DoublyLinkedList<E> in Question 1:

```
import java.util.ArrayList;
public class DoublyLinkedList<E> {

    //---------------- nested Node class ----------------
    /**
     * Node of a doubly linked list, which stores a reference to its
     * element and to both the previous and next node in the list.
     */
    public static class Node<E> {//Note: if you want to make this class private,
you need to write the driver method inside this class
        /** The element stored at this node */
        private E element;               // reference to the element stored at
this node
        /** A reference to the preceding node in the list */
        private Node<E> prev;            // reference to the previous node in the
list
        /** A reference to the subsequent node in the list */
        private Node<E> next;            // reference to the subsequent node in
the list
        /**
         * Creates a node with the given element and next node.
         *
         * @param element  the element to be stored
         * @param prev  reference to a node that should precede the new node
         * @param next  reference to a node that should follow the new node
         */
        public Node(E element, Node<E> prev, Node<E> next) {
            this.element = element;
            this.prev = prev;
            this.next = next;
        }
        // public accessor methods
        /**
         * Returns the element stored at the node.
         * @return the element stored at the node
         */
        public E getElement() { return element; }
        /**
         * Returns the node that precedes this one (or null if no such node).
         * @return the preceding node
         */
        public Node<E> getPrev() { return prev; }
        /**
         * Returns the node that follows this one (or null if no such node).
         * @return the following node
         */
        public Node<E> getNext() { return next; }
        // Update methods
        /**
         * Sets the node's previous reference to point to Node n.
         * @param p    the node that should precede this one
         */
        public void setPrev(Node<E> p) { prev = p; }
        /**
         * Sets the node's next reference to point to Node n.
         * @param n    the node that should follow this one
         */
        public void setNext(Node<E> n) { next = n; }
    } //----------- end of nested Node class -----------

    // instance variables of the DoublyLinkedList
    /** Sentinel node at the beginning of the list */
    private Node<E> header;                    // header sentinel
    /** Sentinel node at the end of the list */
    private Node<E> trailer;                   // trailer sentinel
    /** Number of elements in the list (not including sentinels) */
    private int size = 0;                      // number of elements in the list
    /** Constructs a new empty list. */
    public DoublyLinkedList() {
        header = new Node<>(null, null, null);      // create header
        trailer = new Node<>(null, header, null);   // trailer is preceded by
header
        header.setNext(trailer);                    // header is followed by
trailer
    }
    // public accessor methods
    /**
     * Returns the number of elements in the linked list.
     * @return number of elements in the linked list
     */
    public int size() { return size; }
    /**
     * Tests whether the linked list is empty.
     * @return true if the linked list is empty, false otherwise
     */
    public boolean isEmpty() { return size == 0; }
    /**
     * Adds an element to the end of the list.
     * @param e   the new element to add
     */
    public void addLast(E e) {
        addBetween(e, trailer.getPrev(), trailer);  // place just before the
trailer
    }
    // public update methods
    /**
     * Adds an element to the linked list in between the given nodes.
     * The given predecessor and successor should be neighboring each
     * other prior to the call.
     *
     * @param predecessor   node just before the location where the new element is
inserted
     * @param successor     node just after the location where the new element is
inserted
     */
    public void addBetween(E e, Node<E> predecessor, Node<E> successor) {
        // create and link a new node
        Node<E> newest = new Node<>(e, predecessor, successor);
        predecessor.setNext(newest);
        successor.setPrev(newest);
        size++;
    }
    /**
     * @return the node containing the element e (or null if empty)
     */
    public Node<E> findNode(E e){
        //your code
    }
    /**
     * Produces a string representation of the contents of the list.
     */
    public String toString(){
        //your code
    }
}
```